# Métodos Computacionais para a Engenharia Eletrotécnica

**Ano letivo 2021/2022**

## Ficha Laboratorial 6 – Solução de equações com uma variável

## 1. Objetivo

A presente ficha de trabalho laboratorial tem como objetivo a introdução dos seguintes conceitos:

1. *Toolbox Symbolic Math*.
2. Método da bissecção.
3. Método de Newton-Raphson.
4. Função **`fzero`**.

## 2. Symbolic Math Toolbox

A *Symbolic Math Toolbox* consiste num conjunto de funções destinadas à resolução, obtenção de gráficos e manipulação simbólica de equações matemáticas, com aplicações nas áreas de cálculo, álgebra linear, equações algébricas e diferenciais, e manipulação e simplificação de equações.

### 2.1 Instalação da Toolbox Symbolic

A licença de *campus* disponibilizada permite a instalação e utilização das *toolboxes* desenvolvidas para utilização com o MATLAB, as quais podem ser instaladas aquando da instalação do MATLAB. Caso ainda não tenha instalado a *Symbolic Math Toolbox*, pode sempre adicioná-la posteriormente. Para tal, prima o botão **Add-Ons**, do menu **Home** (Figura 1), o qual abrirá a janela de gestão de extensões (*add-ons*) para o MATLAB. Nesta janela, na barra de pesquisa, no canto superior direito, pode introduzir o termo "symbolic" e premir <*enter*>. Em princípio, o primeiro resultado da pesquisa deverá corresponder ao *Symbolic Math Toolbox* (Figura 2).

Na lista dos resultados da pesquisa, prima na entrada correspondente à *Symbolic Math Toolbox* e, na janela seguinte, prima na opção **Install**. Irá surgir uma mensagem a solicitar a confirmação da instalação, e ainda a informar que o MATLAB será encerrado durante o processo de instalação da *toolbox*. Prossiga a instalação premindo o botão **Continue** e siga as indicações das janelas seguintes de instalação que passa, basicamente, na aceitação da licença.

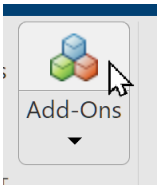


Figura 1. Botão **Add-Ons**.

Em alternativa, para instalar esta *toolbox*, pode executar o programa de instalação do MATLAB. Este vai detetar a instalação do MATLAB existente, e apresentar uma lista de verificação, de onde pode selecionar a(s)

*toolbox(es)* que pretende instalar. Neste caso, deve ter o cuidado de usar o instalador da versão do MATLAB que tem a uso.

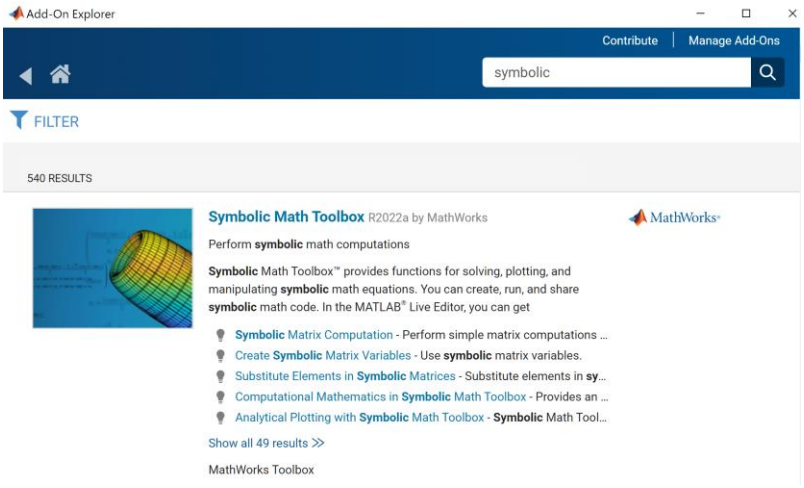


Figura 2. Explorador de extensões (*add-ons*).

## 2.2 Criação e manipulação de variáveis simbólicas

Para a criação de representações simbólicas, podemos recorrer a uma das seguintes funções:

- **`syms`** – esta função cria um objeto simbólico que é automaticamente associado a uma variável com o mesmo nome.
- **`sym`** – esta função *refere-se* a um objeto simbólico que pode ser associado a uma variável com o mesmo nome do objeto simbólico ou não.

Para perceber a diferença entre estas duas funções, considere-se o seguinte exemplo, onde é criada uma variável *x* que contém o objeto simbólico *x:*

```
>> syms x
```

Não há qualquer eco na janela de comando, mas pode verificar facilmente no *workspace* que foi criada uma variável **`x`** do tipo simbólico de dimensão 1x1. A seguir, podemos criar uma variável **`f`** que contém o mesmo objeto simbólico *x*:

```
>> f = sym('x')
f =
x
```

Podemos facilmente verificar que estas duas variáveis, **`x`** e **`f`**, contém o mesmo valor simbólico:

```
>> f + x
ans =
2*x
```

Notar que a representação simbólica pode ser também numérica, i.e., a representação do número sem a aproximação por vírgula-flutuante:

```
>> y = sym('2/3')
y =
2/3
>> z = sym('5/2')
z =
5/2
```

```
>> y + z
ans =
19/6
```

Nas expressões simbólicas, os números são sempre representados na forma de um rácio de dois números inteiros, como se pode ver no último exemplo. A representação 7/12 é a *representação simbólica* deste valor. Para obter a *representação* decimal, podemos usar a função *vpa – variable precision arithmetic*. A sintaxe é **`vpa(<símbolo>, n)`**, onde n corresponde aos números de dígitos pretendidos para a representação:

```
>> h = vpa(y + z, 5)
h =
3.1667
```

Note que este último resultado continua a ser uma *representação simbólica*. Se pretendermos a sua **representação de dupla precisão**, então podemos escrever:

```
>> h = double(h)
h =
    3.1667
```

Neste último caso, o valor de **`h`** é agora um valor numérico.

Considere-se agora o seguinte exemplo:

```
>> syms a b c x
>> y = a*x^2 + b*x + c
y =
a*x^2 + b*x + c
```

É possível avaliar a expressão simbólica **`y`** para valores específicos de **`a`**, **`b`** e **`c`**, com a função **`subs`**, e.g.:

```
>> a = 2; b = 4; c = -1;
>> subs(y)
ans =
2*x^2 + 4*x – 1
```

Notar que, na declaração de y, as variáveis a, b e c não precisam de ser necessariamente simbólicas. Em alternativa, o exemplo anterior poderia ter sido escrito da seguinte maneira:

```
>> a = 2; b = 4; c = -1;
>> syms x
>> y = a*x^2 + b*x + c
y =
2*x^2 + 4*x - 1
```

## 2.3 Diferenciação

A operação de diferenciação simbólica é feita com a função **`diff`**. Considere, por exemplo, a seguinte função de uma variável independente $f(t) = t^2\sin(3t)$. Para determinar $df(t)/dt$, podemos escrever:

```
>> syms t
>> f = t^2*sin(3*t);
>> dfdt = diff(f)
dfdt =
2*t*sin(3*t) + 3*t^2*cos(3*t)
```

A função **diff** permite um segundo argumento numérico que corresponde à ordem da derivada que se pretende determinar. Assim, a segunda derivada de $f(t)$ em ordem a $t$, $d^2f(t)/dt^2$, pode ser obtida da seguinte maneira:

```
>> dfdt2 = diff(f, 2)
dfdt2 =
2*sin(3*t) + 12*t*cos(3*t) - 9*t^2*sin(3*t)
```

Também é possível o cálculo de derivadas parciais com a função **diff**. Neste caso, devemos passar, como argumento da função, qual a variável em ordem à qual se pretende calcular a derivada:

```
>> syms x y
>> g = x*y + log(2*x + 3*y)
g =
log(2*x + 3*y) + x*y

>> diff(g, 'x')
ans =
y + 2/(2*x + 3*y)

>> diff(g, 'y')
ans =
x + 3/(2*x + 3*y)
```

## 2.4 Integração

A função **int** permite a integração definida e indefinida de expressões simbólicas. Por exemplo, para calcular simbolicamente o integral indefinida $\int 5e^{-0.5x}\sin(x)dx$, podemos escrever:

```
>> syms x
>> y = 5*exp(-0.5*x)*sin(x);
>> int(y)
ans =
-4*exp(-x/2)*(cos(x) + sin(x)/2)
```

Para calcular um integral definida da função anterior, por exemplo $\int_0^4 5e^{-0.5x}\sin(x)dx$, basta acrescentar os limites do integral como argumentos de **int**:

```
>> a = int(y, 0, 4)
a =
4 - 2*exp(-2)*(2*cos(4) + sin(4))
>> double(a)
ans =
    4.5587
```

Também é possível o cálculo de integrais impróprias, i.e., um ou ambos os limites tendem para infinito. Neste caso, podemos usar o valor especial **Inf**, o qual corresponde a representação do infinito em MATLAB. Por exemplo:

```
>> double(int(y, 0, Inf))
ans =
     4
```

Finalmente, é possível calcular integrais de funções com duas ou mais variáveis, sendo necessário indicar qual a variável sobre a qual é calculada o integral. Tudo o que foi apresentado sobre integrais indefinidas, definidas e impróprias, mantém-se:

```
>> syms x y
>> h = x*sin(y) + y*exp(-x)
h =
y*exp(-x) + x*sin(y)

>> int(h, 'x')
ans =
(x^2*sin(y))/2 - y*exp(-x)

>> int(h, 'y')
ans =
(y^2*exp(-x))/2 - x*cos(y)

>> vpa(int(h, 'x', 0, pi/3), 3)
ans =
0.649*y + 0.548*sin(y)
```

## 2.5 Conversão de expressões simbólicas em funções anónimas

Depois de manipular uma expressão simbólica, pode existir o interesse em usar explicitamente a expressão como uma função, em vez de ter que recorrer a sequências de funções **`subs`** e **`double`**. Para tal, podemos recorrer à função **`matlabFunction`**:

```
>> syms x
>> y1 = x^2;
>> y2 = sin(x);
>> f = y1*y2;
>> pf = int(f);

>> h = matlabFunction(pf)
h =
  function_handle with value:
    @(x)-cos(x).*(x.^2-2.0)+x.*sin(x).*2.0

>> h(pi/2)
ans =
    3.1416
```

## 2.6 Conversão de funções anónimas em expressões simbólicas

Para converter funções anónimas em expressões simbólicas, basta recorrer à função **`sym`**. As variáveis da função anónima são então referidas como variáveis simbólicas, sem ser necessário qualquer informação explicita.

Considere-se o seguinte exemplo de uma função anónima z(x, y), para a qual pretendemos obter a representação da superfície definida para dz/dx (Figura 3):

```
z = @(x, y) exp(-y)*sin(x);  % função anónima z(x, y)
f = sym(z);                  % converte em expressão simbólica
dfdx = diff(f, 'x');         % derivada em ordem a x, df/dx
dzdx = matlabFunction(dfdx); % de volta a uma função anónima
```

```
% gerar o gráfico da superfície de dz/dx
[X, Y] = meshgrid(-pi:pi/10:pi);
Z = dzdx(X, Y)
surf(X, Y, Z)
xlabel('xx'); ylabel('yy'); zlabel('dzdx(x, y)');
```

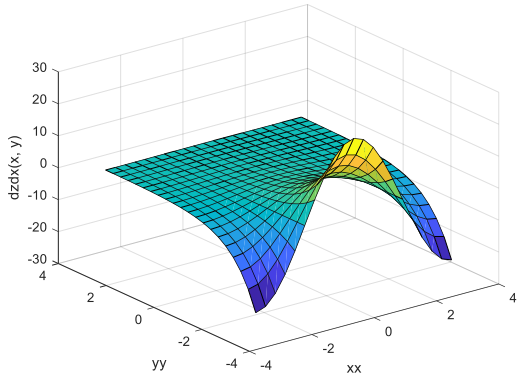


Figura 3. Representação da superfície definida por dz/dx.

## 3. Soluções numéricas para equações com uma variável

Graficamente, uma solução para a função com a forma geral $f(x) = 0$ corresponde ao ponto de intersecção da função $f(x)$ com o eixo dos *xx*. A equação pode ter uma solução, múltiplas soluções ou, simplesmente, não ter qualquer solução, i.e., não há qualquer ponto de intersecção com o eixo dos *xx*.

### 3.1 Método da bissecção

O **Método da Bissecção** é um método numérico fechado para o cálculo da solução de $f(x) = 0$, i.e., requer que seja definido um intervalo $[a, b]$, na qual a função $f(x)$ é contínua e tem uma solução tal que $f(a)\cdot f(b) < 0$.

O algoritmo para determinar a solução da função é o seguinte:

1. Localizar o ponto médio, $c_1$, no intervalo $[a, b]$:

$$c_1 = \frac{1}{2}(a + b)$$

2. Se $f(c_1) = 0$, $c_1$ é solução da equação e termina o algoritmo;
3. Se $f(a)\cdot f(c1) < 0$, a solução está na parte esquerda do intervalo $[a, b]$, e atualizamos o intervalo $[a, b]$ fazendo $b = c_1$;
4. Se $f(a)\cdot f(c1) > 0$, a solução está na parte direita do intervalo $[a, b]$, e atualizamos o intervalo $[a, b]$ fazendo $a = c_1$;
5. Repetimos os passos 1 a 4 até satisfazer um valor de tolerância desejado, $\frac{1}{2}(a - b) < \varepsilon$.

A função **`bisection.m`** é uma possível implementação deste algoritmo. A função recebe a **função f, definida como função anónima**, e os valores $a$ e $b$, que definem os limites do intervalo onde se encontra uma solução de $f$.

```
1     function c = bisection(f, a, b)
2
3     if f(a)*f(b) > 0
4         error('O intervalo não contém nenhuma raiz');
5     end
```

```
6     kmax = 20;          % Define a tolerância (critério de paragem)
7     tol = 1e-4;         % Define a tolerância (critério de paragem)
8     for k = 1:kmax
9         c = (a+b)/2;    % Primeiro ponto médio
10        if f(c) == 0    % Pára se houver uma raíz
11            return
12        end
13
14        if (b-a)/2 < tol  % Valor abaixo da tolerância
15            return
16        end
17
18        if f(b)*f(c) > 0 % Verifica mudança de sinal
19            b = c;       % Ajusta o limite do intervalo
20        else a = c;
21        end
22    end
```

A função termina quando $\frac{1}{2}(a-b) < \varepsilon$, sendo $\varepsilon$ definido no código da função (linha 7). O valor devolvido pela função é o último valor de *c* calculado antes de verificada a condição de paragem.

Note que este algoritmo apenas permite determinar, de cada vez, uma solução da equação. Quando a equação tem várias soluções, é a escolha do intervalo inicial feita pelo utilizador que determina qual a solução a ser determinada. A escolha do intervalo [*a*, *b*], tais que $f(a)\cdot f(b) < 0$, apenas garante que existe, pelo menos, uma solução entre *a* e *b* e, por isso, o método permite a convergência para esse valor.

### 3.2 Método de Newton-Raphson

O **Método de Newton-Raphson** é um método aberto para a resolução de equações do tipo $f(x) = 0$, i.e., requer apenas um ponto de partida para determinar a solução da equação, sem ser necessário definir um intervalo de valores[1].

Seja $x_1$ o ponto de partida para a determinação da solução da equação $f(x) = 0$. A reta tangente no ponto $(x_1, y_1)$, com $y_1 = f(x_1)$, pode ser expressa da seguinte forma:

$$y - f(x_1) = f'(x_1)(x - x_1)$$

onde $f'(x_1)$ é a derivada de $f(x)$ em ordem a *x*. A intersecção desta reta com o eixo dos *xx*, $x_2$, pode ser então determinada como sendo:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

Este novo ponto, assim determinado, é a primeira aproximação à solução da equação. A partir de $x_2$, podemos agora obter a sua imagem $y_2 = f(x_2)$, e a partir deste novo ponto $(x_2, y_2)$, desenhar uma nova reta tangente para obter uma nova intersecção com o eixo dos *xx*, $x_3$, que é:

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

[1] Outros métodos, como o método das secantes, requerem dois pontos de partida.

Se houver convergência, cada novo ponto $x_n$ assim obtido aproximar-se-á da solução de $f(x) = 0$ (Figura 4). O processo é repetido até que dois valores consecutivos estejam suficientemente próximos, i.e., $|x_n - x_{n-1}| < \varepsilon$, sendo $\varepsilon$ a tolerância desejada.

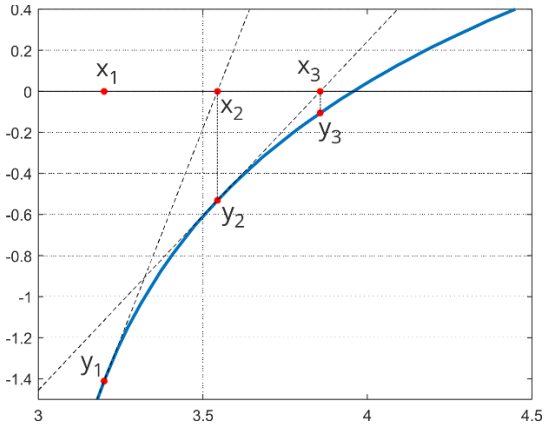

Figura 4. Exemplificação do método Newton-Raphson.

O algoritmo para implementar o método de Newton-Raphson pode ser descrito da seguinte forma:

1. Definir o ponto de partida, $x_1$;
2. Calcular o declive da tangente no ponto ($x_1$, $f(x_1)$);
3. Calcular a próxima solução aproximada, $x_2$;
4. Se $|x_2 - x_1| < \varepsilon$, termina o algoritmo, sendo a solução dada por $x_2$;
5. Se $|x_2 - x_1| > \varepsilon$, fazer $x_1 = x_2$;
6. Repetir os passos 2 a 5 até a solução convergir para a tolerância pretendida.

O método de Newton -Raphson é um método de convergência rápida. No entanto, requer que seja uma função diferenciável. Ainda, a escolha do ponto de partida pode influenciar o comportamento do algoritmo, podendo este não convergir. Por exemplo, se $x_1$ for tal que $f'(x_1) = 0$, o algoritmo é incapaz de determinar o próximo valor de $x_2$.

A função **`newton.m`**, abaixo apresentada, é uma possível implementação do método de Newton-Raphson. Esta função recebe a **função f, definida como função anónima**, e o valor inicial, $x_1$:

```
1     function r = newton(f, x1)
2     % Converte a função anónima em expressão simbólica, calcula a derivada, e
3     % converte o resultado numa nova função anónima
4     y = sym(f);
5     fp = matlabFunction(diff(y));
6
7     tol = 1e-4;  % tolerância
8     N = 30;      % nº máx de iterações (pode ser alterado)
9     for n = 1:N
10        if fp(x1) == 0
11            error('x1 é mín/máx local');
12        end
13        x2 = x1 - f(x1)/fp(x1);
14
15        if abs(x2 - x1) < tol
```

```
16            r = x2;
17            return
18        else
19            x1 = x2;
20        end
22    end
```

### 3.3 Função `fzero`

A função **`fzero`** permite determinar a solução de uma equação da forma $f(x) = 0$ a partir de um valor de $x$ próximo da solução, ou de um intervalo $[a, b]$ especificado, desde que $f(a) \cdot f(b) < 0$.

De forma geral, a função **`fzero`** recebe como argumentos uma função anónima, que corresponde à função para a qual queremos determinar a sua solução, e um escalar (caso se pretenda indicar apenas um valor inicial) ou um vetor com dois valores (caso se pretenda indicar um intervalo):

```
>> f = @(x) (exp(-2*x)-1).*cos(x-pi/3);
>> fzero(f, 5)
ans =
    5.7596

>> fzero(f, [5 6])
ans =
    5.7596

>> % utilização dos algoritmos acima apresentados
>> newton(f, 5)
10  for n = 1:N
ans =
    5.7596

>> bisection(f, 5, 6)
ans =
    5.7596
```

## 4. Atividades

### 4.1 Atividade 1

Escreva um *script* que permita determinar a área sombreada delimitada pelas funções $f(x)$, $g(x)$ e $h(x)$, indicadas na Figura 5. A área deverá ser determinada de forma simbólica e, no final, deverá apresentar uma mensagem com a indicação da área calculada (3 casas decimais).

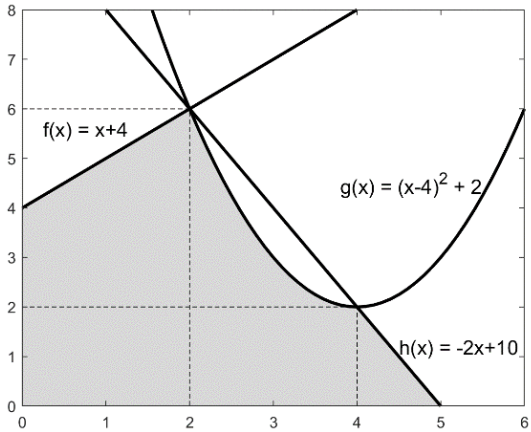


Figura 5. Atividade 1.

## 4.2 Atividade 2

Recorra ao método da bissecção para determinar as soluções de $\sin x \sinh x = \frac{3}{2}$ no intervalo [−2, 2].

a) Altere a função **`bisection.m`** de forma a que esta função devolva não apenas o valor da solução da função, mas também o número de iterações realizadas no cálculo numérico da mesma.

b) Escreva um *script* que permita obter o gráfico da equação e, desse modo, poder identificar graficamente os intervalos que contêm as soluções da equação. Coloque uma grelha sobre o gráfico para facilitar a identificação dos intervalos;

c) No mesmo *script*, use a função **`bisection.m`** para determinar numericamente as soluções da equação, usando como intervalos [*a*, *b*] aqueles que identificou no gráfico da alínea anterior. **Apresente o resultado obtido com seis casas decimais**, e indicando ainda o número de iterações realizados, e.g.,

```
Resultados com bisection.m
Primeira solução, 13 iterações, x1 = -1.241150
Segunda solução,  13 iterações, x2 =  1.241150
```

(*note que os valores apresentados no exemplo podem variar em função dos intervalos [a, b] considerados).*

d) Ainda no mesmo *script*, recorra agora à função **`fzero`**, e volte a calcular as soluções da equação, apresentando novamente os **resultados com seis casa decimais**.

e) Compare os resultados obtidos com **`bisection.m`** e com **`fzero`**. Altere, na função **`bisection.m`**, o valor da tolerância de forma a obter soluções iguais às obtidas com **`fzero`**, até seis casas decimais. Qual foi o custo em termos de número de iterações? E se os resultados forem agora apresentados com 8 casas decimais? Existe ainda muita divergência entre os valores obtidos com **`bisection.m`** e com **`fzero`**?

## 4.3 Atividade 3

Recorra ao método de Newton-Raphson para determinar as soluções da expressão cos*x* cosh*x* = −1.3 no intervalo [−4, 4].

a) Escreva um *script* que permita obter o gráfico da função e, desse modo, poder localizar graficamente as soluções da equação. Coloque uma grelha sobre o gráfico para facilitar a identificação das soluções;

b) Use agora a função **`newton`**, considerando como aproximação a cada solução **o primeiro valor inteiro imediatamente à esquerda de cada solução identificada graficamente**. Verifique os resultados obtidos e compare com os zeros na representação gráfica.

c) Caso algum dos resultados obtidos não seja coerente com as soluções indicadas na representação gráfica, considere, como aproximação à solução, **o valor inteiro imediatamente à direita do zero da função** que não tenha sido calculado corretamente.

d) Como entende a disparidade detetada no cálculo das soluções? E se tivesse considerado, como aproximação à uma das soluções, o valor *x* = 0? E *x* = 0.25?

e) Altere agora o script para permitir o cálculo das mesmas soluções agora com recurso à função **fzero**. Considere as aproximações iniciais tidas nas alíneas b), c) e d), e compare com os resultados obtidos com a função **`newton`**.

### 4.4 Atividade 4

Pretende-se determinar as soluções da função $f(x) = x \sin(x)$ no intervalo de $x \in [-10, 10]$, com recurso à função **`fzero`**. Para tal, escreva um script que realize as seguintes tarefas:

a) Defina uma função anónima para $f(x)$;

b) Divida o intervalo $x \in [-10, 10]$ considerando 20 pontos equidistantes.

c) Calcule o valor de $f(x)$ para os pontos em $x$ considerados na alínea anterior e guarde num vetor;

d) Percorra o vetor de valores obtido na alínea anterior e verifique a condição de mudança de sinal entre cada valor do vetor e o seguinte: se houver mudança de sinal, existe uma solução nesse intervalo.

e) Use a função fzero para calcular a solução dentro do intervalo identificado.

f) Repita as alíneas d) e e) até ter percorrido todos os pontos do intervalo.

g) Apresente as soluções obtidas.

h) De seguida, faça a representação gráfica da função $f(x)$. Note que para ter uma representação gráfica mais exata, terá que ter mais pontos em $x$ do que aqueles que considerou na alínea b).

i) Compare o gráfico obtido com as soluções obtida na alínea g). Estas soluções estão de acordo com o gráfico obtido? Qual o problema que consegue identificar?

## 5. Fora da sala de aula

### 5.1 Exercício 1

Escreva um *script* que permita determinar a área sombreada delimitada pelas funções $f(x)$ e $g(x)$ Figura 6. A área deverá ser determinada de forma simbólica e, no final, deverá apresentar uma mensagem com a indicação da área calculada (3 casas decimais).

Para calcular corretamente a área, deverá determinar os valores de $x$ que correspondem às interseções de $f(x)$ com $g(x)$. Note que estes pontos correspondem a $f(x) = g(x) \Leftrightarrow f(x) - g(x) = 0$. Use a função **`fzero`** para o cálculo destes valores.

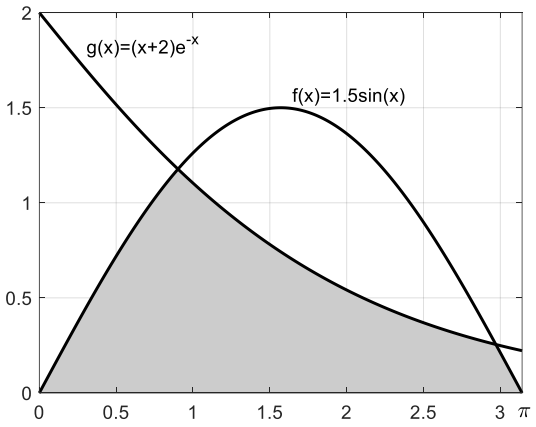


Figura 6. Exercício 1

### 5.2 Exercício 2

Na atividade 2, teve de determinar graficamente os intervalos para o cálculo das soluções da equação proposta. Pretende-se agora que escreva um *script* que permita, de forma automatizada, determinar os intervalos e calcular as soluções da equação da atividade 2.

Para tal, divida em intervalos iguais o intervalo de $x \in [-2, 2]$, e.g., em intervalos de 0.25, e determine quando ocorre a mudança de sinal (*dica: considere usar a função* **`sign`**).

Como poderia fazer isto usando apenas indexação lógica?

## 5.3 Exercício 3

Considere o circuito da Figura 7. A relação tensão corrente de um díodo pode ser descrita por:

$$i_D = I_S \, e^{\frac{q}{kT} v_D} - 1$$

onde $v_D$ e $i_D$ correspondem à tensão e a corrente no díodo, respetivamente, $I_S$ é uma constante que depende da concentração da dopagem do semicondutor, $q = 1.6 \times 10^{-19}$ C é a carga elétrica unitária, $k = 1.38 \times 10^{-23}$ J/K é a constante de Boltzmann, e $T$ é a temperatura absoluta, em kelvin.

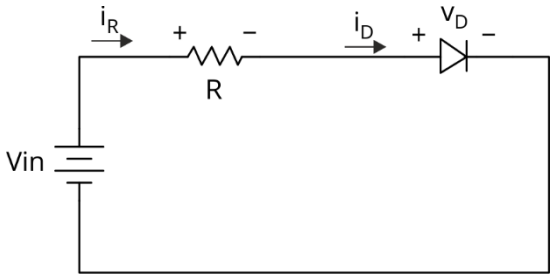


Figura 7. Circuito RD

Aplicando a lei de Kirchhoff das tensões, e sabendo que $i_R = i_D$, podemos escrever:

$$V_{in} - \left( I_S \, e^{\frac{q}{kT} v_D} - 1 \right) R - v_D = 0$$

Sejam $R = 1.5$ kΩ, $T = 300$ K, $I_S = 10^{-14}$ A, e $V_{in} = 5$ V:

a) Crie uma função para $f(v_D)$ de acordo com a expressão atrás obtida, e obtenha o gráfico de $f(v_D)$ com passos de 10 mV, considerando o intervalo $0 \leq v_D \leq 0.8$ V.

b) Determine (**não de forma gráfica!**) um pequeno intervalo onde possa ser encontrada a solução da função $f(v_D)$.

c) Recorra à função fzero para determinar a solução de $f(v_D)$. Apresente a solução na forma de mensagem na janela de comando.